1.1 Grammar Notation

This specification describes two grammars: a lexical grammar and a syntactic grammar. The lexical grammar defines how characters can be combined to form tokens; the syntactic grammar defines how the tokens can be combined to form Visual Basic programs. There are also several secondary grammars used for preprocessing operations like conditional compilation.

> **Note** The grammars in this specification are designed to be human readable, not formal (that is, usable by LEX or YACC).

All of the grammars use a modified BNF notation, which consists of a set of productions made up of terminal and non-terminal names. A terminal name represents one or more Unicode characters. Each nonterminal name is defined by one or more productions. In a production, nonterminal names are shown in *italic type*, and terminal names are shown in a `fixed-width type`. Text in normal type and surrounded by angle-bracket metasymbols are informal terminals (for example, "< all Unicode characters >"). Each grammar starts with the nonterminal *Start*.

Case is unimportant in Visual Basic programs. For simplicity, all terminals will be given in standard casing, but any casing will match them. Terminals that are printable elements of the ASCII character set are represented by their corresponding ASCII characters. Visual Basic is also width insensitive when matching terminals, allowing full-width Unicode characters to match their half-width Unicode equivalents, but only on a whole-token basis. A token will not match if it contains mixed half-width and full-width characters.

A set of productions begins with the name of a nonterminal, followed by two colons and an equal sign. The right side contains a terminal or nonterminal production. A nonterminal may have multiple productions that are separated by the vertical-bar metasymbol (|). Items included in square-bracket metasymbols ([]) are optional. A plus metasymbol (+) following an item means the item may occur one or more times.

Line breaks and indentation may be added for readability and are not part of the production.

13. Grammar Summary

This section summarizes the Visual Basic language grammar. For information on how to read the grammar, see Grammar Notation.

13.1 Lexical Grammar

Start ::= [*LogicalLine+*]

LogicalLine ::= [*LogicalLineElement+*] [*Comment*] *LineTerminator*

LogicalLineElement ::= *WhiteSpace* | *LineContinuation* | *Token*

Token ::= *Identifier* | *Keyword* | *Literal* | *Separator* | *Operator*

13.1.1 Characters and Lines

Character ::= < any Unicode character except a *LineTerminator* >

LineTerminator ::=
 < Unicode carriage return character (0x000D) > |
 < Unicode linefeed character (0x000A) > |
 < Unicode carriage return character > < Unicode linefeed character > |
 < Unicode line separator character (0x2028) > |
 < Unicode paragraph separator character (0x2029) >

LineContinuation ::= *WhiteSpace* `_` [*WhiteSpace+*] *LineTerminator*

WhiteSpace ::=
 < Unicode blank characters (class Zs) > |
 < Unicode tab character (0x0009) >

Comment ::= *CommentMarker* [*Character+*]

CommentMarker ::= *SingleQuoteCharacter* | `REM`

SingleQuoteCharacter ::=
 `'` |
 < Unicode left single-quote character (0x2018) > |
 < Unicode right single-quote character (0x2019) >

13.1.2 Identifiers

Identifier ::=
 NonEscapedIdentifier [*TypeCharacter*] |
 Keyword *TypeCharacter* |
 EscapedIdentifier

NonEscapedIdentifier ::= < *IdentifierName* but not *Keyword* >

EscapedIdentifier ::= `[` *IdentifierName* `]`

IdentifierName ::= *IdentifierStart* [*IdentifierCharacter+*]

IdentifierStart ::=
    AlphaCharacter |
    UnderscoreCharacter *IdentifierCharacter*

IdentifierCharacter ::=
    UnderscoreCharacter |
    AlphaCharacter |
    NumericCharacter |
    CombiningCharacter |
    FormattingCharacter

AlphaCharacter ::=
    < Unicode alphabetic character (classes Lu, Ll, Lt, Lm, Lo, Nl) >

NumericCharacter ::= < Unicode decimal digit character (class Nd) >

CombiningCharacter ::= < Unicode combining character (classes Mn, Mc) >

FormattingCharacter ::= < Unicode formatting character (class Cf) >

UnderscoreCharacter ::= < Unicode connection character (class Pc) >

IdentifierOrKeyword ::= *Identifier* | *Keyword*

TypeCharacter ::=
    IntegerTypeCharacter |
    LongTypeCharacter |
    DecimalTypeCharacter |
    SingleTypeCharacter |
    DoubleTypeCharacter |
    StringTypeCharacter

IntegerTypeCharacter ::= `%`

LongTypeCharacter ::= `&`

DecimalTypeCharacter ::= `@`

SingleTypeCharacter ::= `!`

DoubleTypeCharacter ::= `#`

StringTypeCharacter ::= `$`

13.1.3 Keywords

Keyword ::= < member of keyword table in 2.3 >

13.1.4 Literals

Literal ::=
    BooleanLiteral |
    IntegerLiteral |
    FloatingPointLiteral |
    StringLiteral |
    CharacterLiteral |
    DateLiteral |
    Nothing

BooleanLiteral ::= `True` | `False`

IntegerLiteral ::= *IntegralLiteralValue* [*IntegralTypeCharacter*]

IntegralLiteralValue ::= *IntLiteral* | *HexLiteral* | *OctalLiteral*

IntegralTypeCharacter ::=
  ShortCharacter |
  UnsignedShortCharacter |
  IntegerCharacter |
  UnsignedIntegerCharacter
  LongCharacter |
  UnsignedLongCharacter |
  IntegerTypeCharacter |
  LongTypeCharacter

ShortCharacter ::= `S`

UnsignedShortCharacter ::= `US`

IntegerCharacter ::= `I`

UnsignedIntegerCharacter ::= `UI`

LongCharacter ::= `L`

UnsignedLongCharacter ::= `UL`

IntLiteral ::= *Digit+*

HexLiteral ::= `& H` *HexDigit+*

OctalLiteral ::= `& O` *OctalDigit+*

Digit ::= `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`

HexDigit ::= `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9` | `A` | `B` | `C` | `D` | `E` | `F`

OctalDigit ::= `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7`

FloatingPointLiteral ::=
  FloatingPointLiteralValue [*FloatingPointTypeCharacter*] |
  IntLiteral *FloatingPointTypeCharacter*

FloatingPointTypeCharacter ::=
  SingleCharacter |
  DoubleCharacter |
  DecimalCharacter |
  SingleTypeCharacter |
  DoubleTypeCharacter |
  DecimalTypeCharacter

SingleCharacter ::= `F`

DoubleCharacter ::= `R`

DecimalCharacter ::= `D`

FloatingPointLiteralValue ::=
  IntLiteral `.` *IntLiteral* [*Exponent*] |
  `.` *IntLiteral* [*Exponent*] |
  IntLiteral *Exponent*

Exponent ::= `E` [*Sign*] *IntLiteral*

```
Sign  ::=  +  |  -

StringLiteral  ::=
    DoubleQuoteCharacter  [  StringCharacter+  ]  DoubleQuoteCharacter

DoubleQuoteCharacter  ::=
    "  |
    < Unicode left double-quote character (0x201C) >  |
    < Unicode right double-quote character (0x201D) >

StringCharacter  ::=
    < Character except for DoubleQuoteCharacter >  |
    DoubleQuoteCharacter  DoubleQuoteCharacter

CharacterLiteral  ::=  DoubleQuoteCharacter  StringCharacter  DoubleQuoteCharacter  C

DateLiteral  ::=  #  [  Whitespace+  ]  DateOrTime  [  Whitespace+  ]  #

DateOrTime  ::=
    DateValue  Whitespace+  TimeValue  |
    DateValue  |
    TimeValue

DateValue  ::=
    MonthValue  /  DayValue  /  YearValue  |
    MonthValue  –  DayValue  -  YearValue

TimeValue  ::=
    HourValue  :  MinuteValue  [  :  SecondValue  ]  [  WhiteSpace+  ]  [  AMPM  ]

MonthValue  ::=  IntLiteral

DayValue  ::=  IntLiteral

YearValue  ::=  IntLiteral

HourValue  ::=  IntLiteral

MinuteValue  ::=  IntLiteral

SecondValue  ::=  IntLiteral

AMPM  ::=  AM  |  PM

Nothing  ::=  Nothing

Separator  ::=  (  |  )  |  {  |  }  |  !  |  #  |  ,  |  .  |  :  |  :=

Operator  ::=
    &  |  *  |  +  |  -  |  /  |  \  |  ^  |  <  |  =  |  >  |  <=  |  >=  |  <>  |  <<  |  >>  |
    &=  |  *=  |  +=  |  -=  |  /=  |  \=  |  ^=  |  <<=  |  >>=
```

13.2 Preprocessing Directives

13.2.1 Conditional Compilation

```
Start  ::=  [  CCStatement+  ]

CCStatement  ::=
    CCConstantDeclaration  |
    CCIfGroup |
    LogicalLine

CCExpression ::=
    LiteralExpression |
    CCParenthesizedExpression |
    SimpleNameExpression |
    CCCastExpression |
    CCOperatorExpression

CCParenthesizedExpression ::= ( CCExpression )

CCCastExpression ::= CastTarget ( CCExpression )

CCOperatorExpression ::=
    CCUnaryOperator CCExpression
    CCExpression CCBinaryOperator CCExpression

CCUnaryOperator ::= + | - | Not

CCBinaryOperator ::= + | - | * | / | \ | Mod | ^ | = | <> | < | > |

    <= | >= | & | And | Or | Xor | AndAlso | OrElse | << | >>

CCConstantDeclaration ::= # Const Identifier = CCExpression LineTerminator

CCIfGroup ::=
    # If CCExpression [ Then ] LineTerminator
    [ CCStatement+ ]
    [ CCElseIfGroup+ ]
    [ CCElseGroup ]
    # End If LineTerminator

CCElseIfGroup ::=
    # ElseIf CCExpression [ Then ] LineTerminator
    [ CCStatement+ ]

CCElseGroup ::=
    # Else LineTerminator
    [ CCStatement+ ]
```

13.2.2 External Source Directives

```
Start ::= [ ExternalSourceStatement+ ]

ExternalSourceStatement ::= ExternalSourceGroup | LogicalLine

ExternalSourceGroup ::=
    # ExternalSource ( StringLiteral , IntLiteral ) LineTerminator
    [ LogicalLine+ ]
    # End ExternalSource LineTerminator
```

13.2.3 Region Directives

```
Start ::= [ RegionStatement+ ]

RegionStatement ::= RegionGroup | LogicalLine

RegionGroup ::=
    # Region StringLiteral LineTerminator
    [ LogicalLine+ ]
    # End Region LineTerminator
```

13.2.4 External Checksum Directives

```
Start ::= [ ExternalChecksumStatement+ ]

ExternalChecksumStatement ::=
    # ExternalChecksum ( StringLiteral , StringLiteral , StringLiteral ) LineTerminator
```

13.3 Syntactic Grammar

```
AccessModifier ::= Public | Protected | Friend | Private | Protected Friend

QualifiedIdentifier ::=
    Identifier |
    Global . IdentifierOrKeyword  |
    QualifiedIdentifier . IdentifierOrKeyword

TypeParameterList ::=
    ( Of TypeParameters )

TypeParameters ::=
    TypeParameter |
    TypeParameters , TypeParameter

TypeParameter ::=
    Identifier [ TypeParameterConstraints ]

TypeParameterConstraints ::=
    As Constraint |
    As { ConstraintList }

ConstraintList ::=
    ConstraintList , Constraint |
    Constraint

Constraint ::= TypeName | New
```

13.3.1 Attributes

```
Attributes ::=
    AttributeBlock |
    Attributes AttributeBlock

AttributeBlock ::= < AttributeList >

AttributeList ::=
    Attribute |
    AttributeList , Attribute

Attribute ::=
    [ AttributeModifier : ] SimpleTypeName [ ( [ AttributeArguments ] ) ]

AttributeModifier ::= Assembly | Module

AttributeArguments ::=
    AttributePositionalArgumentList |
```

    AttributePositionalArgumentList `,` *VariablePropertyInitializerList* |
    VariablePropertyInitializerList

AttributePositionalArgumentList ::=
    AttributeArgumentExpression |
    AttributePositionalArgumentList `,` *AttributeArgumentExpression*

VariablePropertyInitializerList ::=
    VariablePropertyInitializer |
    VariablePropertyInitializerList `,` *VariablePropertyInitializer*

VariablePropertyInitializer ::=
    IdentifierOrKeyword `:=` *AttributeArgumentExpression*

AttributeArgumentExpression ::=
    ConstantExpression |
    GetTypeExpression |
    ArrayCreationExpression

13.3.2 Source Files and Namespaces

Start ::=
    [*OptionStatement+*]
    [*ImportsStatement+*]
    [*AttributesStatement+*]
    [*NamespaceMemberDeclaration+*]

StatementTerminator ::= *LineTerminator* | `:`

AttributesStatement ::= *Attributes* *StatementTerminator*

OptionStatement ::=
    OptionExplicitStatement |
    OptionStrictStatement |
    OptionCompareStatement

OptionExplicitStatement ::= `Option Explicit` [*OnOff*] *StatementTerminator*

OnOff ::= `On` | `Off`

OptionStrictStatement ::= `Option Strict` [*OnOff*] *StatementTerminator*

OptionCompareStatement ::= `Option Compare` *CompareOption* *StatementTerminator*

CompareOption ::= `Binary` | `Text`

ImportsStatement ::= `Imports` *ImportsClauses* *StatementTerminator*

ImportsClauses ::=
    ImportsClause |
    ImportsClauses `,` *ImportsClause*

ImportsClause ::= *ImportsAliasClause* | *ImportsNamespaceClause*

ImportsAliasClause ::=
    Identifier `=` *QualifiedIdentifier* |
    Identifier `=` *ConstructedTypeName*

```
ImportsNamespaceClause ::=
    QualifiedIdentifier |
    ConstructedTypeName

NamespaceDeclaration ::=
    Namespace QualifiedIdentifier StatementTerminator
    [ NamespaceMemberDeclaration+ ]
    End Namespace StatementTerminator

NamespaceMemberDeclaration ::=
    NamespaceDeclaration |
    TypeDeclaration

TypeDeclaration ::=
    ModuleDeclaration |
    NonModuleDeclaration

NonModuleDeclaration ::=
    EnumDeclaration |
    StructureDeclaration |
    InterfaceDeclaration |
    ClassDeclaration |
    DelegateDeclaration
```

13.3.3 Types

```
TypeName ::=
    ArrayTypeName |
    NonArrayTypeName

NonArrayTypeName ::=
    SimpleTypeName |
    ConstructedTypeName

SimpleTypeName ::=
    QualifiedIdentifier |
    BuiltInTypeName

BuiltInTypeName ::= Object | PrimitiveTypeName

TypeModifier ::= AccessModifier | Shadows

TypeImplementsClause ::= Implements Implements StatementTerminator

Implements ::=
    NonArrayTypeName |
    Implements , NonArrayTypeName

PrimitiveTypeName ::= NumericTypeName | Boolean | Date | Char | String

NumericTypeName ::= IntegralTypeName | FloatingPointTypeName | Decimal

IntegralTypeName ::= Byte | SByte | UShort | Short | UInteger | Integer | ULong | Long

FloatingPointTypeName ::= Single | Double

EnumDeclaration ::=
    [ Attributes ] [ TypeModifier+ ] Enum Identifier [ As QualifiedName ] StatementTerminator
```

EnumMemberDeclaration+
`End Enum` *StatementTerminator*

EnumMemberDeclaration ::= [*Attributes*] *Identifier* [`=` *ConstantExpression*] *StatementTerminator*

ClassDeclaration ::=
[*Attributes*] [*ClassModifier+*] `Class` *Identifier* [*TypeParameterList*] *StatementTerminator*
[*ClassBase*]
[*TypeImplementsClause+*]
[*ClassMemberDeclaration+*]
`End Class` *StatementTerminator*

ClassModifier ::= *TypeModifier* | `MustInherit` | `NotInheritable` | `Partial`

ClassBase ::= `Inherits` *NonArrayTypeName* *StatementTerminator*

ClassMemberDeclaration ::=
NonModuleDeclaration |
EventMemberDeclaration |
VariableMemberDeclaration |
ConstantMemberDeclaration |
MethodMemberDeclaration |
PropertyMemberDeclaration |
ConstructorMemberDeclaration |
OperatorDeclaration

StructureDeclaration ::=
[*Attributes*] [*StructureModifier+*] `Structure` *Identifier* [*TypeParameterList*]
StatementTerminator
[*TypeImplementsClause+*]
[*StructMemberDeclaration+*]
`End Structure` *StatementTerminator*

StructureModifier ::= *TypeModifier* | `Partial`

StructMemberDeclaration ::=
NonModuleDeclaration |
VariableMemberDeclaration |
ConstantMemberDeclaration |
EventMemberDeclaration |
MethodMemberDeclaration |
PropertyMemberDeclaration |
ConstructorMemberDeclaration |
OperatorDeclaration

ModuleDeclaration ::=
[*Attributes*] [*TypeModifier+*] `Module` *Identifier* *StatementTerminator*
[*ModuleMemberDeclaration+*]
`End Module` *StatementTerminator*

ModuleMemberDeclaration ::=
NonModuleDeclaration |
VariableMemberDeclaration |
ConstantMemberDeclaration |
EventMemberDeclaration |
MethodMemberDeclaration |

```
    PropertyMemberDeclaration  |
    ConstructorMemberDeclaration

InterfaceDeclaration  ::=
    [ Attributes ] [ TypeModifier+ ] Interface Identifier [ TypeParameterList ] StatementTerminator
    [ InterfaceBase+ ]
    [ InterfaceMemberDeclaration+ ]
    End Interface StatementTerminator

InterfaceBase  ::= Inherits InterfaceBases StatementTerminator

InterfaceBases  ::=
    NonArrayTypeName  |
    InterfaceBases , NonArrayTypeName

InterfaceMemberDeclaration  ::=
    NonModuleDeclaration  |
    InterfaceEventMemberDeclaration  |
    InterfaceMethodMemberDeclaration  |
    InterfacePropertyMemberDeclaration

ArrayTypeName  ::= NonArrayTypeName ArrayTypeModifiers

ArrayTypeModifiers  ::= ArrayTypeModifier+

ArrayTypeModifier  ::= ( [ RankList ] )

RankList  ::=
    ,  |
    RankList ,

ArrayNameModifier  ::=
    ArrayTypeModifiers  |
    ArraySizeInitializationModifier

DelegateDeclaration  ::=
    [ Attributes ] [ TypeModifier+ ] Delegate MethodSignature StatementTerminator

MethodSignature  ::= SubSignature | FunctionSignature

ConstructedTypeName  ::=
    QualifiedIdentifier ( Of TypeArgumentList )

TypeArgumentList  ::=
    TypeName  |
    TypeArgumentList , TypeName
```

13.3.4 Type Members

```
ImplementsClause  ::= [ Implements ImplementsList ]

ImplementsList  ::=
    InterfaceMemberSpecifier  |
    ImplementsList , InterfaceMemberSpecifier

InterfaceMemberSpecifier  ::= NonArrayTypeName . IdentifierOrKeyword

MethodMemberDeclaration  ::= MethodDeclaration | ExternalMethodDeclaration

InterfaceMethodMemberDeclaration  ::= InterfaceMethodDeclaration
```

```
MethodDeclaration  ::=
    SubDeclaration  |
    MustOverrideSubDeclaration  |
    FunctionDeclaration  |
    MustOverrideFunctionDeclaration

InterfaceMethodDeclaration  ::=
    InterfaceSubDeclaration  |
    InterfaceFunctionDeclaration

SubSignature  ::=  Identifier  [  TypeParameterList  ]  [  (  [  ParameterList  ]  )  ]

FunctionSignature  ::=  SubSignature  [  As  [  Attributes  ]  TypeName  ]

SubDeclaration  ::=
    [  Attributes  ]  [  ProcedureModifier+  ]  Sub  SubSignature  [  HandlesOrImplements  ]  LineTerminator
    Block
    End Sub  StatementTerminator

MustOverrideSubDeclaration  ::=
    [  Attributes  ]  [  MustOverrideProcedureModifier+  ]  Sub  SubSignature  [  HandlesOrImplements  ]
        StatementTerminator

InterfaceSubDeclaration  ::=
    [  Attributes  ]  [  InterfaceProcedureModifier+  ]  Sub  SubSignature  StatementTerminator

FunctionDeclaration  ::=
    [  Attributes  ]  [  ProcedureModifier+  ]  Function  FunctionSignature  [  HandlesOrImplements  ]
        LineTerminator
    Block
    End Function  StatementTerminator

MustOverrideFunctionDeclaration  ::=
    [  Attributes  ]  [  MustOverrideProcedureModifier+  ]  Function  FunctionSignature
        [  HandlesOrImplements  ]  StatementTerminator

InterfaceFunctionDeclaration  ::=
    [  Attributes  ]  [  InterfaceProcedureModifier+  ]  Function  FunctionSignature  StatementTerminator

ProcedureModifier  ::=
    AccessModifier  |
    Shadows  |
    Shared  |
    Overridable  |
    NotOverridable  |
    Overrides  |
    Overloads

MustOverrideProcedureModifier  ::=  ProcedureModifier  |  MustOverride

InterfaceProcedureModifier  ::=  Shadows  |  Overloads

HandlesOrImplements  ::=  HandlesClause  |  ImplementsClause

ExternalMethodDeclaration  ::=
    ExternalSubDeclaration  |
    ExternalFunctionDeclaration
```

```
ExternalSubDeclaration  ::=
    [ Attributes ] [ ExternalMethodModifier+ ] Declare [ CharsetModifier ] Sub Identifier
        LibraryClause [ AliasClause ] [ ( [ ParameterList ] ) ] StatementTerminator

ExternalFunctionDeclaration  ::=
    [ Attributes ] [ ExternalMethodModifier+ ] Declare [ CharsetModifier ] Function Identifier
        LibraryClause [ AliasClause ] [ ( [ ParameterList ] ) ] [ As [ Attributes ] TypeName ]
        StatementTerminator

ExternalMethodModifier  ::=  AccessModifier | Shadows | Overloads

CharsetModifier  ::=  Ansi | Unicode | Auto

LibraryClause  ::=  Lib StringLiteral

AliasClause  ::=  Alias StringLiteral

ParameterList  ::=
    Parameter |
    ParameterList , Parameter

Parameter  ::=
    [ Attributes ] ParameterModifier+ ParameterIdentifier [ As TypeName ] [ = ConstantExpression ]

ParameterModifier  ::=  ByVal | ByRef | Optional | ParamArray

ParameterIdentifier  ::=  Identifier [ ArrayNameModifier ]

HandlesClause  ::=  [ Handles EventHandlesList ]

EventHandlesList  ::=
    EventMemberSpecifier |
    EventHandlesList , EventMemberSpecifier

EventMemberSpecifier  ::=
    QualifiedIdentifier . IdentifierOrKeyword |
    MyBase . IdentifierOrKeyword |
    Me . IdentifierOrKeyword

ConstructorMemberDeclaration  ::=
    [ Attributes ] [ ConstructorModifier+ ] Sub New [ ( [ ParameterList ] ) ] LineTerminator
    [ Block ]
    End Sub StatementTerminator

ConstructorModifier  ::=  AccessModifier | Shared

EventMemberDeclaration  ::=
    RegularEventMemberDeclaration |
    CustomEventMemberDeclaration

RegularEventMemberDeclaration  ::=
    [ Attributes ] [ EventModifiers+ ] Event Identifier ParametersOrType [ ImplementsClause ]
        StatementTerminator

InterfaceEventMemberDeclaration  ::=
    [ Attributes ] [ InterfaceEventModifiers+ ] Event Identifier ParametersOrType StatementTerminator

ParametersOrType  ::=
    [ ( [ ParameterList ] ) ] |
    As NonArrayTypeName
```

```
EventModifiers ::= AccessModifier | Shadows | Shared

InterfaceEventModifiers ::= Shadows

CustomEventMemberDeclaration ::=
    [ Attributes ] [ EventModifiers+ ] Custom Event Identifier As TypeName [ ImplementsClause ]
        StatementTerminator
        EventAccessorDeclaration+
    End Event StatementTerminator

EventAccessorDeclaration ::=
    AddHandlerDeclaration |
    RemoveHandlerDeclaration |
    RaiseEventDeclaration

AddHandlerDeclaration ::=
    [ Attributes ] AddHandler ( ParameterList ) LineTerminator
    [ Block ]
    End AddHandler StatementTerminator

RemoveHandlerDeclaration ::=
    [ Attributes ] RemoveHandler ( ParameterList ) LineTerminator
    [ Block ]
    End RemoveHandler StatementTerminator

RaiseEventDeclaration ::=
    [ Attributes ] RaiseEvent ( ParameterList ) LineTerminator
    [ Block ]
    End RaiseEvent StatementTerminator

ConstantMemberDeclaration ::=
    [ Attributes ] [ ConstantModifier+ ] Const ConstantDeclarators StatementTerminator

ConstantModifier ::= AccessModifier | Shadows

ConstantDeclarators ::=
    ConstantDeclarator |
    ConstantDeclarators , ConstantDeclarator

ConstantDeclarator ::= Identifier [ As TypeName ] = ConstantExpression StatementTerminator

VariableMemberDeclaration ::=
    [ Attributes ] VariableModifier+ VariableDeclarators StatementTerminator

VariableModifier ::=
    AccessModifier |
    Shadows |
    Shared |
    ReadOnly |
    WithEvents |
    Dim

VariableDeclarators ::=
    VariableDeclarator |
    VariableDeclarators , VariableDeclarator
```

```
VariableDeclarator ::=
    VariableIdentifiers [ As [ New ] TypeName [ ( ArgumentList ) ] ] |
    VariableIdentifier [ As TypeName ] [ = VariableInitializer ]

VariableIdentifiers ::=
    VariableIdentifier |
    VariableIdentifiers , VariableIdentifier

VariableIdentifier ::= Identifier [ ArrayNameModifier ]

VariableInitializer ::= RegularInitializer | ArrayElementInitializer

RegularInitializer ::= Expression

ArraySizeInitializationModifier ::=
    ( BoundList ) [ ArrayTypeModifiers ]

BoundList::=
    Expression |
    0 To Expression |
    UpperBoundList , Expression

ArrayElementInitializer ::= { [ VariableInitializerList ] }

VariableInitializerList ::=
    VariableInitializer |
    VariableInitializerList , VariableInitializer

VariableInitializer ::= Expression | ArrayElementInitializer

PropertyMemberDeclaration ::=
    RegularPropertyMemberDeclaration |
    MustOverridePropertyMemberDeclaration

RegularPropertyMemberDeclaration ::=
    [ Attributes ] [ PropertyModifier+ ] Property FunctionSignature [ ImplementsClause ]
        LineTerminator
    PropertyAccessorDeclaration+
    End Property StatementTerminator

MustOverridePropertyMemberDeclaration ::=
    [ Attributes ] [ MustOverridePropertyModifier+ ] Property FunctionSignature [ ImplementsClause ]
        StatementTerminator

InterfacePropertyMemberDeclaration ::=
    [ Attributes ] [ InterfacePropertyModifier+ ] Property FunctionSignature StatementTerminator

PropertyModifier ::= ProcedureModifier | Default | ReadOnly | WriteOnly

MustOverridePropertyModifier ::= PropertyModifier | MustOverride

InterfacePropertyModifier ::=
    Shadows |
    Overloads |
    Default |
    ReadOnly |
    WriteOnly

PropertyAccessorDeclaration ::= PropertyGetDeclaration | PropertySetDeclaration
```

```
PropertyGetDeclaration ::=
    [ Attributes ] [ AccessModifier ] Get LineTerminator
    [ Block ]
    End Get StatementTerminator

PropertySetDeclaration ::=
    [ Attributes ] [ AccessModifier ] Set [ ( ParameterList ) ] LineTerminator
    [ Block ]
    End Set StatementTerminator

OperatorDeclaration ::=
    UnaryOperatorDeclaration |
    BinaryOperatorDeclaration |
    ConversionOperatorDeclaration

OperatorModifier ::= Public | Shared | Overloads | Shadows

Operand ::= [ ByVal ] Identifier [ As TypeName ]

UnaryOperatorDeclaration ::=
    [ Attributes ] [ OperatorModifier+ ] Operator OverloadableUnaryOperator ( Operand )
        [ As [ Attributes ] TypeName ] LineTerminator
    [ Block ]
    End Operator StatementTerminator

OverloadableUnaryOperator ::= + | - | Not | IsTrue | IsFalse

BinaryOperatorDeclaration ::=
    [ Attributes ] [ OperatorModifier+ ] Operator OverloadableBinaryOperator
        ( Operand , Operand ) [ As [ Attributes ] TypeName ] LineTerminator
    [ Block ]
    End Operator StatementTerminator

OverloadableBinaryOperator ::=
    + | - | * | / | \ | & | Like | Mod | And | Or | Xor |
    ^ | << | >> | = | <> | > | < | >= | <=

ConversionOperatorDeclaration ::=
    [ Attributes ] [ ConversionOperatorModifier+ ] Operator CType ( Operand )
        [ As [ Attributes ] TypeName ] LineTerminator
    [ Block ]
    End Operator StatementTerminator

ConversionOperatorModifier ::= Widening | Narrowing | ConversionModifier
```

13.3.5 Statements

```
Statement ::=
    LabelDeclarationStatement |
    LocalDeclarationStatement |
    WithStatement |
    SyncLockStatement |
    EventStatement |
    AssignmentStatement |
    InvocationStatement |
    ConditionalStatement |
    LoopStatement |
    ErrorHandlingStatement |
    BranchStatement |
    ArrayHandlingStatement |
    UsingStatement

Block ::= [ Statements+ ]

LabelDeclarationStatement ::= LabelName :

LabelName ::= Identifier | IntLiteral

Statements ::=
    [ Statement ] |
    Statements : [ Statement ]

LocalDeclarationStatement ::= LocalModifier VariableDeclarators StatementTerminator

LocalModifier ::= Static | Dim | Const

WithStatement ::=
    With Expression StatementTerminator
    [ Block ]
    End With StatementTerminator

SyncLockStatement ::=
    SyncLock Expression StatementTerminator
    [ Block ]
    End SyncLock StatementTerminator

EventStatement ::=
    RaiseEventStatement |
    AddHandlerStatement |
    RemoveHandlerStatement

RaiseEventStatement ::= RaiseEvent IdentifierOrKeyword [ ( [ ArgumentList ] ) ]
    StatementTerminator

AddHandlerStatement ::= AddHandler Expression , Expression StatementTerminator

RemoveHandlerStatement ::= RemoveHandler Expression , Expression StatementTerminator

AssignmentStatement ::=
    RegularAssignmentStatement |
    CompoundAssignmentStatement |
    MidAssignmentStatement

RegularAssignmentStatement ::= Expression = Expression StatementTerminator

CompoundAssignmentStatement ::= Expression CompoundBinaryOperator Expression StatementTerminator

CompoundBinaryOperator ::= ^= | *= | /= | \= | += | -= | &= | <<= | >>=

MidAssignmentStatement ::=
    Mid [ $ ] ( Expression , Expression [ , Expression ] ) = Expression StatementTerminator

InvocationStatement ::= [ Call ] InvocationExpression StatementTerminator

ConditionalStatement ::= IfStatement | SelectStatement

IfStatement ::= BlockIfStatement | LineIfThenStatement
```

```
BlockIfStatement  ::=
    If  BooleanExpression  [  Then  ]  StatementTerminator
    [  Block  ]
    [  ElseIfStatement+  ]
    [  ElseStatement  ]
    End  If  StatementTerminator

ElseIfStatement  ::=
    ElseIf  BooleanExpression  [  Then  ]  StatementTerminator
    [  Block  ]

ElseStatement  ::=
    Else  StatementTerminator
    [  Block  ]

LineIfThenStatement  ::=
    If  BooleanExpression  Then  Statements  [  Else  Statements  ]  StatementTerminator

SelectStatement  ::=
    Select  [  Case  ]  Expression  StatementTerminator
    [  CaseStatement+  ]
    [  CaseElseStatement  ]
    End  Select  StatementTerminator

CaseStatement  ::=
    Case  CaseClauses  StatementTerminator
    [  Block  ]

CaseClauses  ::=
    CaseClause  |
    CaseClauses  ,  CaseClause

CaseClause  ::=
    [  Is  ]  ComparisonOperator  Expression  |
    Expression  [  To  Expression  ]

ComparisonOperator  ::=  =  |  <>  |  <  |  >  |  =>  |  =<

CaseElseStatement  ::=
    Case  Else  StatementTerminator
    [  Block  ]

LoopStatement  ::=
    WhileStatement  |
    DoLoopStatement  |
    ForStatement  |
    ForEachStatement

WhileStatement  ::=
    While  BooleanExpression  StatementTerminator
    [  Block  ]
    End  While  StatementTerminator

DoLoopStatement  ::=  DoTopLoopStatement  |  DoBottomLoopStatement

DoTopLoopStatement  ::=
    Do  [  WhileOrUntil  BooleanExpression  ]  StatementTerminator
    [ Block ]
    Loop StatementTerminator

DoBottomLoopStatement ::=
    Do StatementTerminator
    [ Block ]
    Loop WhileOrUntil BooleanExpression StatementTerminator

WhileOrUntil ::= While | Until

ForStatement ::=
    For LoopControlVariable = Expression To Expression [ Step Expression ] StatementTerminator
    [ Block ]
    Next [ NextExpressionList ] StatementTerminator

LoopControlVariable ::=
    Identifier [ ArrayNameModifier ] As TypeName |
    Expression

NextExpressionList ::=
    Expression |
    NextExpressionList , Expression

ForEachStatement ::=
    For Each LoopControlVariable In Expression StatementTerminator
    [ Block ]
    Next [Expression ] StatementTerminator

ErrorHandlingStatement ::=
    StructuredErrorStatement |
    UnstructuredErrorStatement

StructuredErrorStatement ::=
    ThrowStatement |
    TryStatement

TryStatement ::=
    Try StatementTerminator
    [ Block ]
    [ CatchStatement+ ]
    [ FinallyStatement ]
    End Try StatementTerminator

FinallyStatement ::=
    Finally StatementTerminator
    [ Block ]

CatchStatement ::=
    Catch [ Identifier As NonArrayTypeName ] [ When BooleanExpression ] StatementTerminator
    [ Block ]

ThrowStatement ::= Throw [ Expression ] StatementTerminator

UnstructuredErrorStatement ::=
    ErrorStatement |
    OnErrorStatement |
    ResumeStatement
```

ErrorStatement ::= `Error` *Expression* *StatementTerminator*

OnErrorStatement ::= `On Error` *ErrorClause* *StatementTerminator*

ErrorClause ::=
 `GoTo - 1` |
 `GoTo 0` |
 GotoStatement |
 `Resume Next`

ResumeStatement ::= `Resume` [*ResumeClause*] *StatementTerminator*

ResumeClause ::= `Next` | *LabelName*

BranchStatement ::=
 GotoStatement |
 ExitStatement |
 ContinueStatement |
 StopStatement |
 EndStatement |
 ReturnStatement

GotoStatement ::= `GoTo` *LabelName* *StatementTerminator*

ExitStatement ::= `Exit` *ExitKind* *StatementTerminator*

ExitKind ::= `Do` | `For` | `While` | `Select` | `Sub` | `Function` | `Property` | `Try`

ContinueStatement ::= `Continue` *ContinueKind* *StatementTerminator*

ContinueKind ::= `Do` | `For` | `While`

StopStatement ::= `Stop` *StatementTerminator*

EndStatement ::= `End` *StatementTerminator*

ReturnStatement ::= `Return` [*Expression*]

ArrayHandlingStatement ::=
 RedimStatement |
 EraseStatement

RedimStatement ::= `ReDim` [`Preserve`] *RedimClauses* *StatementTerminator*

RedimClauses ::=
 RedimClause |
 RedimClauses `,` *RedimClause*

RedimClause ::= *Expression* *ArraySizeInitializationModifier*

EraseStatement ::= `Erase` *EraseExpressions* *StatementTerminator*

EraseExpressions ::=
 Expression |
 EraseExpressions `,` *Expression*

UsingStatement ::=
 `Using` *UsingResources* *StatementTerminator*
  [*Block*]
 `End Using` *StatementTerminator*

UsingResources ::= *VariableDeclarators* | *Expression*

13.3.6 Expressions

```
Expression ::=
    SimpleExpression |
    TypeExpression |
    MemberAccessExpression |
    DictionaryAccessExpression |
    IndexExpression |
    NewExpression |
    CastExpression |
    OperatorExpression

ConstantExpression ::= Expression

SimpleExpression ::=
    LiteralExpression |
    ParenthesizedExpression |
    InstanceExpression |
    SimpleNameExpression |
    AddressOfExpression

LiteralExpression ::= Literal

ParenthesizedExpression ::= ( Expression )

InstanceExpression ::= Me

SimpleNameExpression ::= Identifier [ ( Of TypeArgumentList ) ]

AddressOfExpression ::= AddressOf Expression

TypeExpression ::=
    GetTypeExpression |
    TypeOfIsExpression |
    IsExpression

GetTypeExpression ::= GetType ( GetTypeTypeName )

GetTypeTypeName ::=
    TypeName |
    QualifiedIdentifier ( Of [ TypeArityList ] )

TypeArityList ::=
    , |
    TypeParameterList ,

TypeOfIsExpression ::= TypeOf Expression Is TypeName

IsExpression ::=
    Expression Is Expression |
    Expression IsNot Expression

MemberAccessExpression ::=
    [ [ MemberAccessBase ] . ] IdentifierOrKeyword

MemberAccessBase ::=
    Expression |
    BuiltInTypeName |
    Global |
    MyClass |
    MyBase

DictionaryAccessExpression ::= [ Expression ] ! IdentifierOrKeyword

InvocationExpression ::= Expression [ ( [ ArgumentList ] ) ]

ArgumentList ::=
    PositionalArgumentList , NamedArgumentList |
    PositionalArgumentList |
    NamedArgumentList

PositionalArgumentList ::=
    Expression |
    PositionalArgumentList , [ Expression ]

NamedArgumentList ::=
    IdentifierOrKeyword := Expression |
    NamedArgumentList , IdentifierOrKeyword := Expression

IndexExpression ::= Expression ( [ ArgumentList ] )

NewExpression ::=
    ObjectCreationExpression |
    ArrayCreationExpression |
    DelegateCreationExpression

ObjectCreationExpression ::=
    New NonArrayTypeName [ ( [ ArgumentList ] ) ]

ArrayCreationExpression ::=
    New NonArrayTypeName ArraySizeInitializationModifier ArrayElementInitializer

DelegateCreationExpression ::= New NonArrayTypeName ( Expression )

CastExpression ::=
    DirectCast ( Expression , TypeName ) |
    TryCast ( Expression , TypeName ) |
    CType ( Expression , TypeName ) |
    CastTarget ( Expression )

CastTarget ::=
    CBool | CByte | CChar | CDate | CDec | CDbl | CInt | CLng | CObj | CSByte | CShort |
    CSng | CStr | CUInt | CULng | CUShort

OperatorExpression ::=
    ArithmeticOperatorExpression |
    RelationalOperatorExpression |
    LikeOperatorExpression |
    ConcatenationOperatorExpression |
    ShortCircuitLogicalOperatorExpression |
    LogicalOperatorExpression |
    ShiftOperatorExpression

ArithmeticOperatorExpression ::=
    UnaryPlusExpression |
    UnaryMinusExpression |
    AdditionOperatorExpression |
    SubtractionOperatorExpression  |
    MultiplicationOperatorExpression  |
    DivisionOperatorExpression  |
    ModuloOperatorExpression  |
    ExponentOperatorExpression

UnaryPlusExpression  ::=  +  Expression

UnaryMinusExpression  ::=  -  Expression

AdditionOperatorExpression  ::=  Expression  +  Expression

SubtractionOperatorExpression  ::=  Expression  -  Expression

MultiplicationOperatorExpression  ::=  Expression  *  Expression

DivisionOperatorExpression  ::=
    FPDivisionOperatorExpression  |
    IntegerDivisionOperatorExpression

FPDivisionOperatorExpression  ::=  Expression  /  Expression

IntegerDivisionOperatorExpression  ::=  Expression  \  Expression

ModuloOperatorExpression  ::=  Expression  Mod  Expression

ExponentOperatorExpression  ::=  Expression  ^  Expression

RelationalOperatorExpression  ::=
    Expression  =  Expression  |
    Expression  <>  Expression  |
    Expression  <  Expression  |
    Expression  >  Expression  |
    Expression  <=  Expression  |
    Expression  >=  Expression

LikeOperatorExpression  ::=  Expression  Like  Expression

ConcatenationOperatorExpression  ::=  Expression  &  Expression

LogicalOperatorExpression  ::=
    Not  Expression  |
    Expression  And  Expression  |
    Expression  Or  Expression  |
    Expression  Xor  Expression

ShortCircuitLogicalOperatorExpression  ::=
    Expression  AndAlso  Expression  |
    Expression  OrElse  Expression

ShiftOperatorExpression  ::=
    Expression  <<  Expression  |
    Expression  >>  Expression

BooleanExpression  ::=  Expression
```